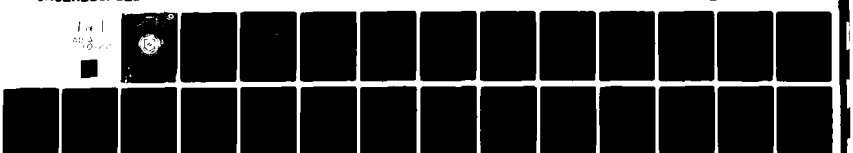
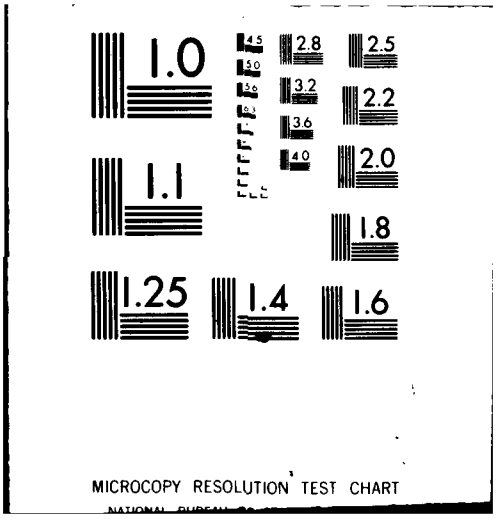
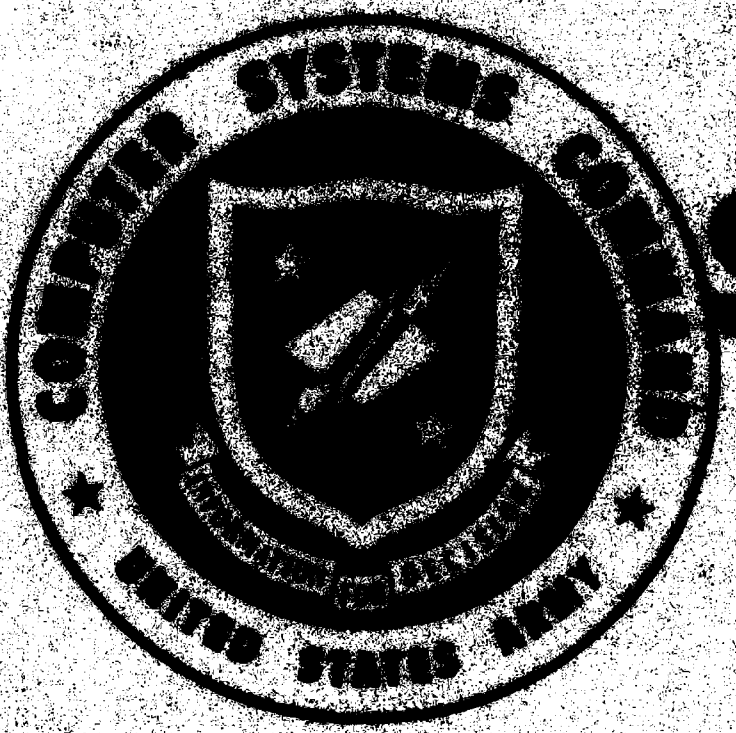
AD-A110 122 GEORGIA INST OF TECH ATLANTA F/G 5/5
COMPUTER AIDED SPECIFICATION TESTING SYSTEM (CASTS), VOLUME III--ETC(U)
MAR 81 D E HUMPHREY, D P MILLARD DAAK70-79-D-0087
UNCLASSIFIED NL



END
DATE FILMED
2-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART



CASTS
COMPUTER AIDED SPECIFICATION
TESTING SYSTEM
VOLUME III - ORDAL PROGRAMMING CONVENTIONS
WORD 2000

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A1120	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CASTS (COBOL PROGRAMMING CONVENTIONS)		5. TYPE OF REPORT & PERIOD COVERED R&D Technical Report Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) D. E. Humphrey, D. P. Millard EES, Georgia Institute of Technology		8. CONTRACT OR GRANT NUMBER(s) DAAK70-79-D-0087 Task Order Number 0007
9. PERFORMING ORGANIZATION NAME AND ADDRESS Georgia Institute of Technology Atlanta, Georgia 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Institute for Research in Management Information and Computer Science 115 O'Keefe Bldg., GIT Atlanta, Georgia 30332		12. REPORT DATE 15 March 1981 13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE None

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Aided Design, Man-Machine Interface, Human Factors, Specification Testing, Interactive, Data Entry.

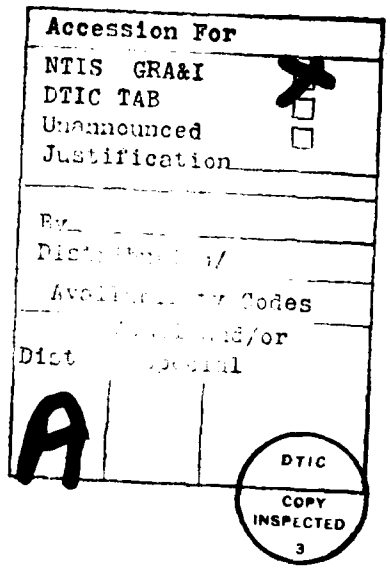
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Computer Aided Specifications Testing System (CASTS) is a tool to aid in the design of interactive systems. It is concerned with human factors and the man-machine interface during the initial capture of data and the editing of that data. CASTS is divided into two processes. The first process is that used by the designer to construct simulations of interactive video display terminal dialogue; the second process gathers the performance data of an actual user who runs the simulation designed in the first process.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

153800

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



COBOL PROGRAMMING CONVENTIONS

Prepared by
Computer Science and Technology Laboratory
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, Georgia 30332

D. E. Humphrey

AIRMICS
Georgia Institute of Technology
Room 325, Hinman Research Building
Atlanta, Georgia 30332

Under DAAK70-79-D-0087, Task Order 0007

TABLE OF CONTENTS

Supersession/Update Information:

This is a new document, Version 0, May 29, 1980.

0.0 INTRODUCTION

Motivations for the design of the programming language, COBOL, included achieving a common standard language and improving the readability of source programs. COBOL has certainly enjoyed widespread application, but the opportunities for clarity and readability of programs are often neglected. This document specifies a set of programming conventions, emphasizing COBOL coding practices, for use in writing programs which are easy to understand and to maintain.

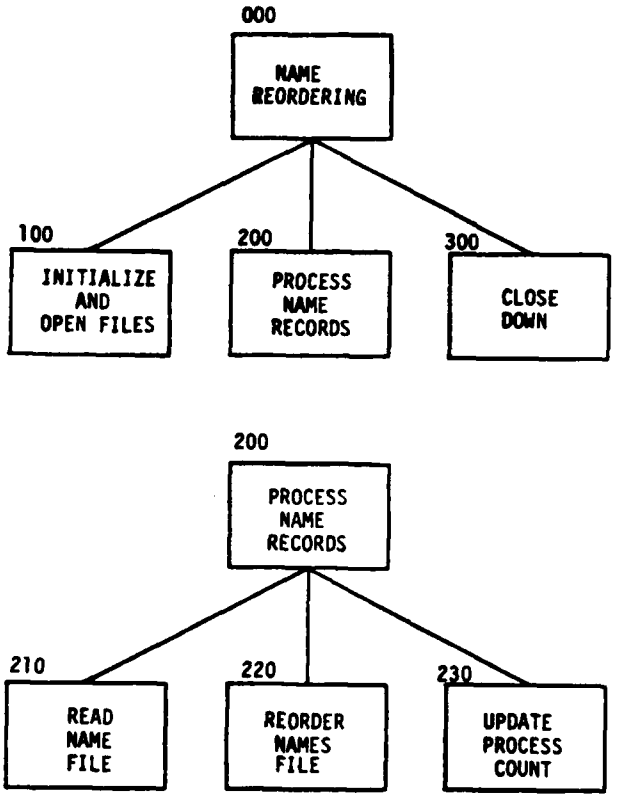
The programming conventions described herein attempt to embody the concepts of structured programming, consistency, understandability and readability. Stepwise refinement and top-down development go hand in hand with structured programming and modularity. They should be reflected in both the final coded program and the guidelines for coding that program. Thus, the conventions include such things as a consistent indentation scheme, the use of meaningful data and paragraph names and the avoidance of nonstandard features.

1.0 PROGRAMMING CONSIDERATIONS

1.1 Notes on Program Design

The following notes should be taken into consideration during the design phase of program development. The idea to keep in mind is clarity. Say what is meant, simply and directly. Generally, all programs should be structured so that they are readable, logically efficient and easily maintained. Choose a data representation that makes the program simple. Modularize the program structure. Each module should do one thing well. Bad code should be rewritten, not patched.

Top-down structured program design lends itself to a hierarchical program structure. The suggested approach is to consider the overview of the program function as a "zero level" routine (or paragraph). This corresponds to the root of a tree. Consider each function in turn. On a VTOC (Visual Table of Contents) diagram this would be graphically organized from left to right. At each level, each function can be further broken down into its components until a primitive function level is reached. (see diagram)



Example VTOC

Input and output should be planned carefully. Test input for validity and plausibility. Ensure that input cannot violate program limits or cause an abnormal ending. Terminate input by an end-of-file mark, not by a count. Identify bad input and recover if possible. Finally, localize input and output in order to facilitate debugging and future maintenance.

Code runs faster and is easier to read and understand if it is compact. Replace repetitive expressions by a paragraph that may be performed when needed. However, don't strain to re-use code; reorganize instead. Make sure special cases are truly special. Keep the program simple to make it faster. Make it right before making it faster. Keep it right when making it faster. Don't sacrifice clarity for small gains in "efficiency".

Documentation is very important in providing maintainability. In-line documentation means more than just a few comments. It can be the most useful form of documentation with minimal effort. First, make sure comments and code agree. Do not just echo the code, make comments count. Don't comment bad code; rewrite it. And don't over comment. In-line documentation also entails using meaningful variable names and paragraph names, indenting to indicate logical program structure and formatting to help the reader understand the program.

1.2 Terminal Format vs. Conventional Format

All of the guidelines for conventional format apply equally well to terminal format pro-

grams. Only two differences need be pointed out. First, since there are no columns for sequencing in terminal format, all conventions referring to column 8 translate to column 1, and likewise column 12 to column 5. Second, the comment indicator (*) should be placed in column 1.

2.0 CONVENTIONS THROUGHOUT A PROGRAM

2.1 Comment lines should be used in all sections where a concept might need explanation other than what is evident in the COBOL code. Normally, however, well chosen data and paragraph names should convey the meaning.

2.2 Comment lines should be used to document calls to external routines.

2.3 Division headers and paragraph headers begin in column 8.

2.4 Indentation: successive levels of indentation will consist of 4 spaces.

2.5 Line spacing: space 3 blank lines before a division header, 2 lines before a section header, 1 line before a paragraph header, 1 line between group items in the Data Division and no line after the paragraph header and before the paragraph body.

2.6 Although words may be broken off and continued on the next line, this should be avoided for the sake of readability.

3.0 IDENTIFICATION DIVISION

3.1 Should contain:
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
REMARKS.

3.2 PROGRAM-ID is the first sentence and should be preceded by 2 blank lines. The name given as the program identifier is the same as the source file name.

3.3 The AUTHOR is the programmer who originally coded the program.

3.4 The DATE-WRITTEN is the date coding began.

3.5 DATE-COMPILED signals the compiler to print the date on the program listing.

3.6 The REMARKS paragraph.

3.6.1 Form.

3.6.1.1 All REMARKS lines have an asterisk (*) in column 7.

3.6.1.2 Place each remark on its own line, indenting any continuation lines 4 spaces.

3.6.1.3 Three types of remarks are 1) functional, 2) additional and 3) modificational. Each group is separated by one blank line (* in column 7), with no blank lines among remarks within a group.

3.6.2 Functional remarks appear first and contain the application or project name and a brief description of what the program does.

3.6.3 Additional remarks are second, if appropriate, and include such things as subroutines referenced, library modules referenced (COPY or CALL modules) and program switches.

3.6.4 Program modification remarks.

3.6.4.1 Each modification line has the form: *MOD* YY/MM/DD - text line, with indented continuation lines.

3.6.4.2 These remarks assist in tracking down maintenance debug errors, providing modification history documentation. They should be brief and concise.

4.0 ENVIRONMENT DIVISION

4.1 Leave 3 blank lines between the Identification Division and the Environment Division header and 2 blank lines after the header, before the Configuration Section.

4.2 Division header, section names and paragraphs begin in column 8.

4.3 Should contain:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
    SOURCE-COMPUTER.
    OBJECT-COMPUTER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    <file SELECT statements>
```

4.4 SELECT statements.

4.4.1 Each SELECT statement is separated from the previous statement by one blank line.

4.4.2 Place the SELECT and ASSIGN clauses on the same line whenever possible, beginning in column 12.

4.4.3 Place additional clauses on separate lines, indenting each line to column 16.

4.4.4 Suggested order for SELECT statements is according to order of usage of the files within the program. An alternative ordering is listing the most active files first.

5.0 DATA DIVISION

5.1 Leave 3 blank lines between the Environment Division and the Data Division header and 2 blank lines after the header, before the first section.

5.2 Division header, section names, paragraphs, FDs and level 01 data descriptions begin in column 8.

5.3 FD declarations.

5.3.1 Precede FDs with one blank line.

5.3.2 Leave 2 spaces between "FD" and its file name.

5.3.3 FDs should appear in the same order as their corresponding SELECT statements.

5.3.4 Additional FD clauses begin on separate lines, indented to column 12.

5.3.5 Those file descriptions that are permanent or are used more than once within a system of programs should be stored in a library and copied into the programs.

5.4 Data description level numbers.

5.4.1 Precede all level 01 numbers with one blank line.

5.4.2 Level numbers subordinate to 01 are assigned in increments of 1 (except 88).

5.4.3 Leave 2 spaces between a level number and its data name.

5.4.4 Level 01 descriptions begin in column 8. Successive levels are indented 4 spaces. After level 02, indenting only 2 spaces is allowed if and only if multiple level spacing becomes a problem.

5.5 Align all PICTURE clauses in one column. Align VALUE and USAGE clauses as much as possible.

5.5.1 The VALUE clause should appear on the same line as the PICTURE clause, if it is the first clause following the PICTURE clause.

If the value to be assigned is twelve characters long or less, it should also go on the same line. Longer values continue on the next line indented the standard 4 spaces.

5.6 Use meaningful data names. (see 4.10.3 also)

EX. RANGE-BOUNDS
REPORT-TITLE

5.7 Use PIC X. instead of PIC A. for greater update flexibility.

5.8 When defining constants with lengthy VALUE literals such as headings, use several elementary data items to sub-define the item. This will simplify maintenance, particularly when a report must be adjusted to add an element or align the heading.

5.9 Use FILLER for any data item not explicitly referenced, unless the item description is in a COPY module.

5.10 Working Storage Section.

5.10.1 Level 77 items are not allowed, use 01 items.

5.10.2 Variables with similar functions should be grouped together under level 01 data names.

EX.
CONSTANTS	POINTERS
COUNTERS	SWITCHES
FLAGS	any tables
MESSAGES	

5.10.3 Naming conventions for level 01 groups.

5.10.3.1 Counters should end in "-CTR" or "-CNT".

5.10.3.2 Message data items should have "-MSG" appended to them.

5.10.3.3 Pointers should end in "-PTR".

5.10.3.4 Switches should end in "-SW" and should have level 88 condition names assigned for testing.

```
EX.  02  EOF-SW   PIC 9   VALUE 0.
         88  EOF           VALUE 1.
```

5.10.4 All pointers, switches, keys, etc. should be initialized.

5.10.5 Switches use 1 for "true" or "on" and 0 for "false" or "off".

6.0 PROCEDURE DIVISION

6.1 Leave 3 blank lines between the Data Division and the Procedure Division header and 2 blank lines after the header, before the first section.

6.2 The division header, section names and paragraph names begin in column 8.

6.3 There should be no GOTO statements! There is only one exception; in the INPUT and/or OUTPUT PROCEDURE sections of a SORT, and then there should be only one to branch to the exit paragraph.

6.4 Modular Program Structure.

6.4.1 A module refers to a unit of code that: 1) has one entry point, 2) has one exit point, 3) has one function and 4) can be referenced by an identifier as a unit. In other words a module is usually a paragraph, but can also be construed as a section or subroutine.

6.4.2 A program should contain:

Main line routine
Initialization routine
Processing modules
Input/Output routines
End of job routine

6.4.2.1 The main line routine should be the highest level executive controlling paragraph for the program. It causes all other paragraphs to be performed and it should have little or no conditional logic.

6.4.2.2 The initialization takes care of the "setting up" functions, such as opening files, getting the date, initializing variables, etc.

6.4.2.3 Processing modules should each perform one function, reflected by the module name.

6.4.2.4 Input/Output routines contain all reads, writes and rewrites for

each file and are grouped together. There should be only one paragraph in the program for each of these functions. Maintenance is simplified by centralizing direct I/O operations.

6.4.2.5 The end of job routine should perform closing "housekeeping" functions, such as closing files, processing last records and end of report routines.

6.4.3 Module Naming Conventions.

6.4.3.1 Each module (paragraph or section) name consists of two parts: 1) a 3 digit number, 2) a meaningful name. This is helpful from the program design phase through testing and debugging, documenting and finally maintenance.

6.4.3.2 The digit portion of a module name is indicative of the module's position, both logically in the program hierarchy and physically in the coded program. Modules are arranged in ascending numerical order, with the controlling main routine being "000". They should reflect a tree-like structure with levels.

6.4.3.3 Within the same program, verbs used in module names ought to have the same meaning.

6.5 One blank line separates each paragraph body from the following header.

6.6 Each statement should appear on a separate line, no multiple statement lines. This enhances readability and update ease.

6.7 Four spaces are used for logic level indentation.

6.8 Statement formats.

6.8.1 If a statement is longer than one line, continue on successive lines at the start of a clause or phrase, indenting 4 spaces.

```
EX.  PERFORM <paragraph name>
         UNTIL <condition1>
             AND <condition2>.
```

6.8.2 In statements such as OPEN, CLOSE and sometimes MOVE, similar elements should be aligned, i.e. file and data names.

```
EX.  OPEN INPUT  OLD-MASTER-FILE
                 UPDATE-FILE
          OUTPUT NEW-MASTER-FILE
                 ERROR-REPORT-FILE.

EX.  MOVE ZERO      TO LINE-CNT.
     MOVE SPACES    TO PRINT-LINE.
     MOVE PART-INFO TO PRINT-PART.
```

6.8.3 IF statements.

6.8.3.1 Put the IF and ELSE (if there is one) on separate lines from the indented statements.

```
EX.  IF <condition>
         <statement>
     ELSE
         <statement>.
```

6.8.3.2 Nested IF statements are also indented. If there are more than 5 levels of nested IFs, re-evaluate the program design. The levels may be indented only 2 spaces if necessary to keep from running out of coding space.

6.8.3.3 For the IF statement used in a "case" structure, the next IF should appear on the same line as the ELSE and all ELSEs should be aligned.

```
EX.  IF <condition1>
         <statement1>
     ELSE IF <condition2>
         <statement2>
     ELSE
         <statement3>.
```

6.9 Compound conditions.

6.9.1 In compound conditions, use parentheses to indicate the order of evaluation.

6.9.2 Align similar parts.

```
EX.  IF <condition1>
        OR <condition2>
        OR (<condition3> AND <condition4>)
           <statement>.
```

6.10 Often used literals should be defined as constants.

6.11 If a specific occurrence of a table item is used repeatedly, move it to an unsubscripted data element and use that data element instead.

6.12 Files should be opened and closed immediately before and after use. Multiple opening and closing of a file should be avoided.

6.13 In programs that use the CALL statement, use as few data names as possible in the USING list. A level 01 can represent all arguments with individual items appearing as subdefinitions, which can be initialized as required prior to the CALL.

6.14 All programs should be impervious to input data.

6.14.1 Check divisors for a zero value.

6.14.2 Range checks should be performed on input used as table subscripts. The range limits should be defined as constants in Working Storage, not literals because table size may change through program maintenance.

6.14.3 Items used in calculations should be tested for NUMERIC and range before use.

6.14.4 Error handling procedures should indicate the nature of the errors that might occur.

6.15 Restrict the use of COBOL INDEXes and SET statements since these are inflexible and may hinder maintenance.

APPENDIX A Acknowledgements

The need for some sort of standardization of COBOL programs, in spite of various programmers and their diverse backgrounds, provided the incentive for producing the COBOL Programming Conventions. The effort was based on the documents entitled "Computer Operations Group, COBOL Program Conventions" (May 7, 1980) by Thomas S. Brinks of the Price Gilbert Memorial Library, GIT, and "COBOL Program Standards" (May 1979), Office of the Registrar and OCS Applications System Design, GIT. Suggestions and corrections were offered by EES/CSTL associates David Winters, Gary Peckham and Joe Celko.

This document was initially developed under the CASTS Project A-2560, funded by AIRMICS, GIT. These conventions are adhered to throughout the CASTS system software in an effort to provide consistency and maintainability.

APPENDIX B Example Program

```
IDENTIFICATION DIVISION.

PROGRAM-ID.                         EXAMPLE.
AUTHOR.                             DAVID WINTERS.
DATE-WRITTEN.                       28-MAY-1980.
DATE-COMPILED.
REMARKS.                            NAME REORDERING.
*
* THIS PROGRAM READS FROM THE FILE 'INPUT' THE
*      NAME TO BE RE-ORDERED AND CONVERTS IT TO THE
*      FORMAT (LAST NAME, FIRST NAME, MIDDLE NAME)
*      AND WRITES IT TO FILE "OUTPUT".


ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
    SOURCE-COMPUTER.                VAX-11.
    OBJECT-COMPUTER.                VAX-11.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

    SELECT NAME-FILE-IN ASSIGN TO "INPUT".

    SELECT NAME-FILE-OUT ASSIGN TO "OUTPUT".

    SELECT PRINTER ASSIGN TO "PRINT".


DATA DIVISION.

FILE SECTION.

FD  NAME-FILE-IN
    LABEL RECORDS ARE OMITTED
    DATA RECORDS IS NAME-RECORD-IN.

01  NAME-RECORD-IN.
    02  FIRST-NAME-IN              PIC X(10).
    02  MIDDLE-NAME-IN             PIC X.
```

```
    02  LAST-NAME-IN              PIC X(20).

FD  NAME-FILE-OUT
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS NAME-RECORD-OUT.

01  NAME-RECORD-OUT.
    02  LAST-NAME-OUT             PIC X(20).
    02  FIRST-NAME-OUT            PIC X(10).
    02  MIDDLE-NAME-OUT           PIC X.

FD  PRINTER
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRINT-LINE.

01  PRINT-LINE.
    02  FILLER                    PIC X(133).


WORKING-STORAGE SECTION.

01  FILE-CONTROL-SWITCHES.
    02  EOF-INPUT-SW              PIC 9      VALUE 0.
        88 EOF                               VALUE 1.
        88 NOT-EOF                           VALUE 0.

01  RECORDS-PROCESSED-COUNTERS.
    02  RECORDS-READ              PIC 9999   VALUE 0.
    02  RECORDS-WRITTEN           PIC 9999   VALUE 0.

01  PRINTER-MESSAGES.
    02  PROCESSED-MSG.
        03  FILLER                PIC X      VALUE SPACE.
        03  FILLER                PIC X(5)   VALUE
            "READ ".
        03  RECORDS-READ-OUT      PIC ZZZZ.
        03  FILLER                PIC X(5)   VALUE SPACES.
        03  FILLER                PIC X(8)   VALUE
            "WRITTEN ".
        03  RECORDS-WRITTEN-OUT   PIC ZZZZ.


**************************************************************

PROCEDURE DIVISION.


MAIN-PROGRAM SECTION.

000-MAIN-LINE.
    PERFORM 100-INITIALIZE-AND-OPEN.
    PERFORM 200-PROCESS-NAME-RECORDS
```

```
           UNTIL EOF.
       PERFORM 300-CLOSE-DOWN.
       STOP RUN.

   100-INITIALIZE-AND-OPEN.
       OPEN INPUT NAME-FILE-IN
            OUTPUT NAME-FILE-OUT, PRINTER.

   200-PROCESS-NAME-RECORDS.
       PERFORM 210-READ-NAME-FILE.
       IF NOT-EOF
           PERFORM 220-REORDER-NAMES-FILE
           PERFORM 230-UPDATE-PROCESS-COUNT.

   210-READ-NAME-FILE.
       READ NAME-FILE-IN AT END
           MOVE 1 TO EOF-INPUT-SW.

   220-REORDER-NAMES-FILE.
       MOVE LAST-NAME-IN TO LAST-NAME-OUT.
       MOVE FIRST-NAME-IN TO FIRST-NAME-OUT.
       MOVE MIDDLE-NAME-IN TO MIDDLE-NAME-OUT.
       WRITE NAME-RECORD-OUT.

   230-UPDATE-PROCESS-COUNT.
       ADD 1 TO RECORDS-READ.
       ADD 1 TO RECORDS-WRITTEN.

   300-CLOSE-DOWN.
       MOVE PROCESSED-MSG TO PRINT-LINE.
       WRITE PRINT-LINE.
       CLOSE NAME-FILE-IN, NAME-FILE-OUT, PRINTER.
```

INDEX